Accelerating software engineering research adoption with Analysis Bots

Ivan Beschastnikh
University of British Columbia

Mircea F. Lungu
University of Groningen

Yanyan Zhuang
University of Colorado, Colorado Springs

***Abstract*—An important part of software engineering (SE) research is to develop new analysis techniques and to integrate these techniques into software development practice. However, since access to developers is non-trivial and research tool adoption is slow, new analyses are typically evaluated as follows: a prototype tool that embeds the analysis is implemented, a set of projects is identified, their revisions are selected, and the tool is run in a controlled environment, rarely involving the developers of the software. As a result, research artifacts are brittle and it is unclear if an analysis tool would actually be adopted.**

In this paper, we envision harnessing the rich interfaces provided by popular social coding platforms for automated deployment and evaluation of SE research analysis. We propose that SE analyses can be deployed as *analysis bots*. We focus on two specific benefits of such an approach: (1) analysis bots can help evaluate analysis techniques in a less controlled, and more realistic context, and (2) analysis bots provide an interface for developers to "subscribe" to new research techniques without needing to trust the implementation, the developer of the new tool, or to install the analysis tool locally. We outline basic requirements for an analysis bots platform, and present research challenges that would need to be resolved for bots to flourish.

I. Introduction

One important objective of the SE research community is to develop new analyses to help software practitioners build more robust, correct, and efficient software systems. Unfortunately, few researchers are motivated to maintain their analysis implementations after publication; and fewer still actively engage practitioners. This slows down adoption of research ideas, and also hinders scientific replication as other researchers struggle to build on prior work [14], [16].

On the other hand, social coding platforms like GitHub and BitBucket are accumulating software projects and developers at a staggering rate. The influence of these platforms is such that many developers now list their social coding platform accounts on their resume to showcase their portfolio [15].

Source code centralization and a powerful set of APIs in social coding platforms have resulted in numerous *tools* to automate or augment SE tasks[1]. One example, Requires.io[2] monitors dependencies of Python projects on GitHub and alerts developers when one of the project's dependencies is outdated, particularly in the case of security updates; Another example, Reviewable.io[3], provides an advanced code review interface on top of GitHub's existing stream of commits and pull requests. In this paper, we consider one way in which we can leverage this emerging category of tools to accelerate the adoption of *SE research* analysis techniques.

We envision an ecosystem of *analysis bots* that effectively extend a single researcher's ability to impact software practitioners. An analysis bot is an automated software agent that (1) helps to automate a useful SE task, and (2) interacts with the software project that it analyzes through a well-defined interface. For example, a bot may implement a program repair analysis and submit a pull request that contains a bug fix [12]. Or, a bot could recommend in the issue comments the developer who should resolve the bug [7], or post an estimate of how long the issue would take to be resolved [20]. Bots could also propose improvements to other project resources like documentation [19]. In all of these cases, the analysis bots partly automate an SE task. An analysis bot can analyze a variety of project artifacts, including the source code, issue tracker information, wiki pages, pull requests, and more. Therefore, we use the term *analysis* broadly and take it to mean more than just classic software analysis.

The examples above suggest a number of challenges that must be addressed for analysis bots to become a reality. What are the appropriate and inappropriate ways in which these bots can interact with developers? How should analysis bots be designed to prevent information overload for developers? How and where should these bots be hosted? We propose, as one way to realize the vision of analysis bots and to answer these questions, a *platform* that mediates the interaction between project hosting sites, such as GitHub, and analysis bots. The focus of this paper, therefore, is on sketching out some key properties of such a platform, which we call *Mediam*.

We have two objectives for Mediam. First, we want it to help researchers evaluate their techniques by lowering the barrier to the adoption of new analysis techniques. Second, we want to facilitate software developers experimenting with state-of-the-art analysis techniques without the overhead associated with finding analysis implementations, or configuring and installing these locally. We believe that this will create more early adopters and will accelerate SE innovation adoption.

Structure of the paper. Section II discusses the goals of the envisioned platform. Section III introduces the workflow for the platform from the perspective of three different stakeholders. Section IV discusses design considerations and Section V highlights the key challenges that must be addressed for the vision to succeed. Section VI discusses the broader implications of the proposed infrastructure and Section VII discusses related work.

[1]https://github.com/integrations
[2]https://requires.io/
[3]https://reviewable.io/

II. Design goals

Mediam mediates between those who develop analyses, such as SE researchers, and software developers. It hosts *analysis bots*, which bot creators (e.g., researchers) publish, and which developers associate with their projects. Mediam has the following set of design goals:

Low barrier to entry. Analysis bots in Mediam are run in the cloud and do not need to be installed/hosted/monitored. It suffices for developers to register their cloud hosted project with Mediam. At the same time, there must be a low barrier to entry for those who want to build bots. For this, Mediam must have a set of well-defined APIs for bots to use when dealing with project resources, and for Mediam to interact with bots.

Fine-grained bot control. Developers must be able to select the analysis bots (or the types of analysis) that interest them and choose the projects to be analyzed. The developers must also be in control of all of the ways in which Mediam can interact with their projects — resources available to bots, what bots can and cannot modify in the project, etc.

Openness. For our vision to become a reality, we believe that anyone should be able to build an analysis bot. We do not want to limit this capability because we cannot predict the bots that would be most useful to developers. One implication of openness is that we need to help developers find and select among the available bots. One way to do this is with a *BotStore*, where analysis bots register, receive reviews from SE developers, and have a reputation score.

III. Platform workflow

There are three types of stake-holders in Mediam: bot creators, maintainers who monitor Mediam and its repository of available bots, and software developers who work on SE projects in social coding platforms. In this section we describe several scenarios of how we envision analysis bots can function from the perspective of different stake holders.

Crista, the bot creator is a software engineering researcher who has written many software analysis tools for Python. To accompany one of her papers focused on software engineering education she creates an analysis bot dubbed *CodeLikeGuido*, which automatically detects non-idiomatic Python. She submits the new bot for deployment on Mediam.

Her analysis implementation has two parts: a pattern detector, which looks for typical beginner mistakes in source code; and a recommendation generator, which sends a pull request together with a link to a page describing the reason why the change is recommended. Her motivation is to help others and to have her tool be quickly adopted by others by using the exposure that Mediam provides.

Manfred, the Mediam maintainer is part of a volunteer team that maintains Mediam. Today he receives a notification about a new bot called *CodeLikeGuido*, implemented by Crista. Manfred and his team run a battery of tests and inspect some of the bot's code to make sure that no malicious code is included. Since Crista has published other highly-used bots in the past, the bot is accepted.

Manfred publishes *CodeLikeGuido* in the *test pool* – a group of bots whose recommendations are initially shown to a limited set of users, the first adopters. Their feedback is closely observed to further vet a new bot.

Dean, the developer is a beginner software developer who is starting a new NLP project in Python. After hearing about analysis bots that verify code and provide free code improvement advice, he decides to register his project with GitHub and Mediam. He selects the option that allows his project to be found by the bots (the other option is to search for relevant bots himself).

Once registered with Mediam, several Python bots including *CodeLikeGuido* start monitoring his project. One day, when Dean pushes his changes to GitHub, he receives a pull request from Mediam which provides several refactorings that will make his code more *pythonic*. Together with the pull request there is a link that explains the reasoning and advantages of the new style. He reads the explanation, agrees with it, and accepts the pull request. Observing that the pull request has been accepted, Mediam adjusts upwards the *reputation* of *CodeLikeGuido*. It also increases a *relevance score* for *CodeLikeGuido* with respect to Dean.

IV. Architecture

Figure 1 presents a high level view of Mediam design. It presents two of the main stake-holder groups: bot creators and developers. Mediam provides a set of APIs that these stake-holders use to interact with the various resources in the system.

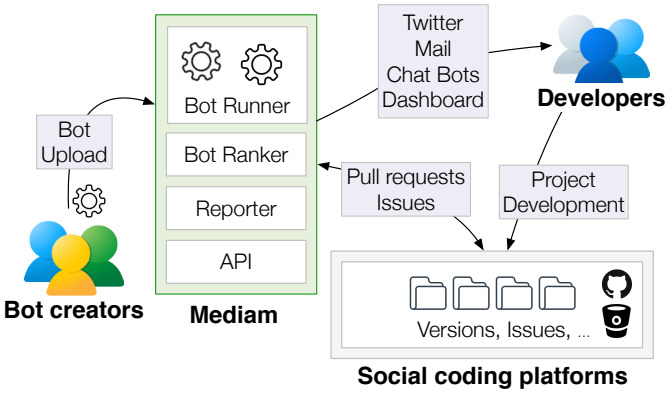

Fig. 1. A high-level view of the Mediam architecture.

Bot creators create bots for different SE tasks and submit bots to the bot repository. Executing bots are hosted in virtual machines or Docker containers (*Bot Runner* in the Figure). The Mediam maintainers monitor the bot repository and all executing bots. Submitted bots undergo a review, during which the bot is tested. Bots that pass the review are published to developers who can enable them for a project (or they are recommended the most relevant bots). The bots communicate with SE projects and developers using Mediam APIs. Bots may generate a variety of events; these are filtered and aggregated by the *Reporter* module (which in turn is helped by the *Bot Ranker* module) before being passed back to developers.

Bot submission API allows bot creators to submit their bots for review. The bot creator later receives an email from the maintainers with the bot review result. If accepted, the creator will also receive a secret key with which she can access the rest of the Mediam API. Submitting a follow-on version of a bot does not require a full review, but each new version must be signed with the same private key as the initial version.

Bot repository API. Mediam maintainers use the bot repository API to review and approve bots. Additionally, SE developers can use this API to review the bot according to several criteria, such as utility and usability.

Project evolution API. Mediam uses hooks to inform a bot about events that the bot subscribes to for a particular project. Events may be new commits to the repository, changes to the team associated with a project, pull requests, issue submissions, and other project changes. Mediam provides a unified view of these events through a single API that hides the differences among social coding platforms.

Project resources API. Analysis bots use this API to obtain data about SE projects that they analyze. This API is constrained by policies that the SE developer sets up ahead of time to limit bot access to certain resources. For open source projects, we expect that bots would have access to all of the project's data. In some cases, SE developers may expose additional resources to bots. For example, a developer may allow the bot to notify the project team through a chat bot in Slack or Basecamp.

Bot reporting API allows a bot to communicate with Mediam and to provide feedback to the project that it analyzed. This API allows a bot to specify a preferred feedback channel (email, pull request, chat bot, etc). A bot can also annotate recommendations with an importance level. However, Mediam will compute its own relevance score based on observations of developer reactions to past recommendations by this bot.

V. Challenges

Realizing the analysis bots vision will require resolving several challenges; we discuss the most important ones below.

Trust. Mediam must be able to host and execute, potentially malicious, analysis bot code. Furthermore, the framework must monitor bot activity and suspend bots that attempt to circumvent bot restrictions. We expect this challenge to be much less severe on the developer side, since the bots do not execute on a developer's machine, and most bot actions will be reversible and can always be inspected.

Bot discovery. An implication of the openness goal (Section II) is that there may be a large number of bots that a developer may select from. Mediam will help developers find the right bots using one of two alternatives:

- A *BotStore* can list analysis bots that can be reviewed and rated. Crowdsourcing bot certification will naturally identify the bots that are most useful to the community.
- A model of a bot's reputation and utility can be inferred using machine learning. This can be done by observing user-bot interactions. New bots can be gradually recommended to developers to bootstrap this process.

Information overload. We expect that most analysis bots will generate a variety of reports, issues, pull requests, notifications on social media channels, or other intermediate project artifacts that developers would need to review and act on. To avoid overwhelming developers with output and false positives [13], [8], it is up to Mediam to decide which notifications are sent to the developers. This indirection is critical to avoid information overload. At the same time, Mediam must ensure fairness towards all bots: established bots should not prevent new ones from being heard.

Bot debugging. Scaling analyses to real and diverse software projects is a major challenge for bot creators [8]. Mediam must help creators to identify and quickly fix issues with their bots, particularly because the reputation of the entire Mediam ecosystem is on the line.

VI. Discussion

Implications for SE research. We believe that analysis bots will benefit the SE research community in two ways: (1) they will help evaluate analysis techniques in a less controlled, and more realistic context. The usage-based evaluation that analysis bots provide could allow a researcher to argue that their analysis is useful because of, for example, the number of pull requests that have been accepted by actual developers from the corresponding analysis bot. More generally we envision Mediam collecting a variety of telemetry information to help bot creators understand bot usage. And, (2) they will lower the bar for developers trying new, state-of-the-art software analysis techniques since analysis bots provide an interface for developers to "subscribe" to research tools without needing to trust the tool or tool developers, or to install the tool locally.

Motivations for building bots. We imagine that analysis bots would be primarily implemented by SE researchers who are usually eager to share their ideas with the world. However, the Mediam framework may encourage others to contribute new analysis as well. Several notable projects, such as Wikipedia and StackOverflow, have relied on volunteer contributors. Although analysis bots require more expertise than either of these efforts, we believe that there are many software engineers who would be interested in contributing to analysis bots if their value and impact is clearly articulated. Another motive for bot authors is the reputation that they would build for themselves through their bot creations.

Hosting analysis bots. The Mediam platform can initially host bots. However, a more sustainable approach is to let bot creators run the bots themselves and then report the results of analysis to Mediam. Another model for hosting bots is the approach used by volunteer computing platforms like Folding@Home and Seti@Home [6]. These platforms use spare cycles from volunteer machines around the world for their computation. Mediam could likewise farm out analysis bot computations to volunteer computers around the world.

Beyond social coding sites. Integrating with social coding sites, like GitHub, will help to bootstrap the platform. However, by building on distributed version control systems like

git, Mediam can pull directly from any code server and can be eventually used by non-open-source projects, as well.

Is this even SE research? Many of the challenges of creating and running Mediam are engineering concerns. However, there are fundamental SE research questions underlying the platform: how can SE practitioners be induced to evolve their practices, and in particular to try new tools? Which analyses are most valuable/useful, and how does the answer change between projects and developers?

VII. Related work

Bots that interact with software projects are not a new idea. For example, the Code Drones vision has some similarities with analysis bots, though drones are clearly much more capable [5]. A more concrete example is the Imageoptimiser bot[4], which automatically generates a pull request with more optimized images. The Hubot[5] project provides a set of APIs for developing *interactive* chat bots. A number of Hubot-based bots exist for automating SE tasks. In a recent paper, Storey and Zagalsky proposed a framework for evaluating how bots are used in software development [18].

Bots have also been used extensively in online communities, such as Wikipedia [10], [4] and Reddit [1]. On these sites, bots automate repetitive and simple tasks, like the ClueBot_NG[6], which detects and reverts vandalism edits on Wikipedia.

The Imageoptimiser bot noted above was rebuked by GitHub, which has indicated that this kind of bot activity is unwelcome [2]. We believe that a platform like Mediam is necessary to protect developers from spam and to help to develop a coherent set of policies around automated agents in the GitHub ecosystem.

Several platforms have been previously proposed that are similar to our analysis bots vision in that they propose to automate and scale software analysis across a large number of applications. For example, Testing-as-a-service [9] was proposed by Candea et al. to provide an automated software testing platform for use by developers and users. Another example is AppInspector [11], which was proposed as a tool for automated mobile app analysis and security validation, with the goal of analyzing all apps in existing centralized app markets. Both platforms focus on a specific type of analysis, interact only minimally with the analyzed software project or app, and are not intended for extensions and contributions from third parties.

Google's internal version control system receives the *majority* of commits from automated bots [3] and the type of contributions these bots perform varies broadly.

Recently, Tricorder, Google's software analysis platform has been detailed by Sadowski et al. [17]. This platform has been open sourced as ShipShape[7]. Both Tricorder and ShipShape include many features that are important to the Mediam platform in the longer term. Technically, the key difference is that these systems are designed to be used in-house while we envision Mediam to be more distributed.

[4]https://github.com/imageoptimiser

[5]https://hubot.github.com/

[6]http://en.wikipedia.org/wiki/User:ClueBot_NG

[7]https://github.com/google/shipshape

VIII. Conclusion

SE researchers often struggle with evaluating their research proposals since accessing real developers can be nontrivial. Many SE researchers have turned to using online open source resources available through hosting sites like GitHub. But, even when evaluated against real code, the value of an analysis technique to an actual developer is often unclear.

We described Mediam, a platform for hosting *analysis bots*, which are automated analysis agents that perform useful SE tasks. These bots can be implemented by anyone, but are especially useful to researchers who can use analysis bots to more easily evaluate their proposals. We believe that analysis bots have the potential to accelerating SE research adoption by practitioners. We are currently prototyping Mediam and plan to evaluate the resulting platform in our future work.

References

[1] faq - botwatch. http://www.reddit.com/r/botwatch/wiki/faq. Accessed October 11, 2016.

[2] GitHub Says No Thanks to Bots Even if Theyre Nice. http://www.wired.com/2012/12/github-bots/. Accessed October 11, 2016.

[3] Google Is 2 Billion Lines of Code — And It's All in One Place. http://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/. Accessed October 5, 2016.

[4] Wikipedia:Bots - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Wikipedia:Bots. Accessed October 11, 2016.

[5] M. P. Acharya, C. Parnin, N. A. Kraft, A. Dagnino, and X. Qu. Code Drones. In *ICSE*, 2016.

[6] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@Home: An Experiment in Public-resource Computing. *CACM*, 2002.

[7] J. Anvik, L. Hiew, and G. C. Murphy. Who Should Fix This Bug? In *ICSE*, 2006.

[8] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World. *CACM*, 2010.

[9] G. Candea, S. Bucur, and C. Zamfir. Automated Software Testing As a Service. In *SoCC*, 2010.

[10] R. S. Geiger. The Lives of Bots. *Wikipedia: A Critical Point of View. Amsterdam: Institute of Network Cultures*, 2001.

[11] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung. Vision: Automated Security Validation of Mobile Apps at App Markets. MCS, 2011.

[12] C. L. Goues, T. Nguyen, S. Forrest, and W. Weimer. GenProg: A Generic Method for Automatic Software Repair. *TSE*, 38(1):54–72, 2012.

[13] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? In *ICSE*, 2013.

[14] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary Guidelines for Empirical Research in Software Engineering. *TSE*, 28(8):721–734, Aug. 2002.

[15] J. Marlow and L. Dabbish. Activity Traces and Signals in Software Developer Recruitment and Hiring. In *CSCW*, 2013.

[16] M. Runeson, Perand Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, 2008.

[17] C. Sadowski, J. van Gogh, C. Jaspan, E. Soederberg, and C. Winter. Tricorder: Building a Program Analysis Ecosystem. In *ICSE*, 2015.

[18] M.-A. Storey and A. Zagalsky. Disrupting Developer Productivity One Bot at a Time. In *FSE*, 2016.

[19] C. Treude and M. P. Robillard. Augmenting API Documentation with Insights from Stack Overflow. In *ICSE*, 2016.

[20] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How Long Will It Take to Fix This Bug? In *MSR*, 2007.